TP 2 : Théorie des graphes

Instructions. Il s'agit d'un TP Python pour lequel il est conseillé d'utiliser Spyder Python 3.6. Voici les principales consignes :

- Un espace de dépôt est ouvert sur Célène, vous devez y déposer votre travail avant la date prévue.
- Le rendu doit être un fichier .zip contenant : un compte rendu contenant les réponses aux questions ainsi que les programmes Python (les fichiers .py).
- Vous pouvez travailler en binôme.

1 Introduction

Dans ce TP nous allons utiliser plusieurs bibliothèques bien connues en Python afin de représenter des graphes. On commencera donc par ajouter en début de script les lignes suivantes :

```
import networkx as nx #Cette bibliothèque permettra de manipuler des graphes
import matplotlib.pyplot as plt #Cette bibliothèque permettra de les représenter
import numpy as np
```

Manipulation de graphes à l'aide du package networkx

Le but de cette section est de vous apprendre à définir et à manipuler des graphes à l'aide du package **networkx**. Néanmoins, si besoin, je vous invite par exemple à visiter ce site où à mener vos propres recherches pour vous familiariser avec cette bibliothèque. Voici donc quelques opérations simples sur les graphes accessibles grâce à cette bibliothèque.

Créer un graphe :

```
G = nx.Graph() #Crée un graphe G non orienté
G2= nx.DiGraph() #Crée un graphe G orienté
```

Ajouter/supprimer de sommets :

```
G.add_node(1) #Ajoute le sommet 1
G.add_nodes_from([2,3]) #Ajoute les éléments de la liste comme sommets
G.remove_node(1) #Supprime le sommet 1
G.remove_nodes_from([1,2]) #Supprime les éléments de la liste comme sommet
```

Ajouter/supprimer d'arrêtes :

```
G.add_edge(1,2) #Ajoute l'arrête 1-2 (et ajoute le sommet s'il n'existe pas)
G.add_edge(2,4,weight=4) #Ajoute l'arrête 2-4 avec une pondération de 4
G[2][4]['weight'] #Retourne le poids de l'arrête 2-4
G.add_edges_from([(1,2),(1,3)]) #Ajoute les arrêtes à partir de la liste
G.add_weighted_edges_from([(1,2,0.125),(1,3,0.75),(2,4,1.2),(3,4,0.375)])
#Ajoute les arrêtes pondérées à partir de la liste
G.remove_edge(1,3) #Supprime l'arrête 1-3
G.remove_edges_from([(1,2),(0,1)]) #Supprime les arrêtes à partir de la liste
```

Représenter un graphe :

```
nx.draw(G) #Représente le graphe G en plaçant les sommets de façon aléatoire
plt.show() #Utile surtout si vous tracez plusieurs graphes dans un seul script
```

Charger un graphe depuis un fichier :

```
GG=nx.read_adjlist("./test.txt") #Charge le graphe à partir du fichier test.txt
nx.draw(GG)
nx.write_adjlist(G,"test2.txt") #Sauve le graphe G dans le fichier test2.txt
```

où le fichier text.txt est un fichier texte de la forme

```
1 2 3 4
2 5
```

où le premier nombre de la ligne est le nom du sommet et ensuite ce sont les noms de ses voisins qui sont renseignés.

Exemples de fonctions utiles :

```
G.number_of_nodes() #Nombre de sommets de G
G.number_of_edges() #Nombre d'arrêtes de G
G.nodes() #Liste des sommets de G
G.edges() #Liste des arrêtes de G
```

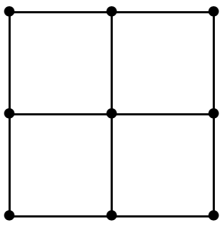
Exercice 1

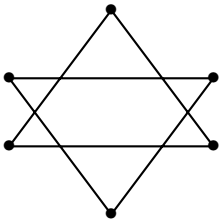
Le but de ce premier exercice est simplement de mettre en œuvre les différentes notions introduites précédemment :

1. Créer un graphe G et un graphe orienté DG.
2. Ajouter à chaque graphe les sommets nommés 1, 2, 3, 4 et 5.
3. Afficher les sommets des graphes.
4. Supprimer le sommet 1 du graphe G.
5. Ajouter au graphe G les arêtes $\{2,3\}$, $\{2,5\}$, $\{3,4\}$ et $\{4,5\}$.
6. Ajouter au graphe DG les arêtes $(1,3)$, $(2,3)$, $(2,4)$, $(2,5)$, $(4,5)$ et $(5,1)$.
7. Tracer le graphe G et le graphe DG. Quelles sont les limites d'un tel affichage ?

Exercice 2

Représenter les graphes suivants :





Exercice 3

Écrire un script qui crée un graphe non orienté dont :

1. Le nombre de sommets est choisi aléatoirement entre 2 et 15.
2. L'existence d'une arrête entre deux sommets existants est aléatoire.

Indication : On pourra utiliser la fonction **random** du package **random**.

Améliorations possibles de la représentation graphique

Afin d'obtenir une représentation graphique plus lisible des graphes, la bibliothèque **networkx** permet d'utiliser d'avantage d'options comme la coloration d'un nœud, l'ajout d'une étiquette à un nœud où encore l'ajout du poids à une arrête.

Déclaration du graphe :

```
import matplotlib.pyplot as plt
import networkx as nx
from numpy import array
# Définition du graphe
G = nx.Graph()
# Définition des noeuds
G.add_node(0,label='A',col='pink')
G.add_node(1,label='B',col='red')
G.add_node(2,label='C',col='white')
G.add_node(3,label='D',col='white')
G.add_node(4,label='E',col='white')
G.add_node(5,label='F',col='blue')
# Définition des aretes
G.add_edge(0,1,weight=6)
G.add_edge(0,2,weight=5)
G.add_edge(0,4,weight=1)
```

```
G.add_edge(4,1,weight=5)
G.add_edge(4,2,weight=1)
G.add_edge(4,3,weight=3)
G.add_edge(2,3,weight=8)
G.add_edge(4,5,weight=6)
G.add_edge(3,5,weight=9)
# Représentation du graphe basique
nx.draw(G)
plt.show()
```

Traitement des nœuds :

```
# Colorer les noeuds :
liste = list(G.nodes(data='col'))
colorNodes = {}
for noeud in liste:
    colorNodes[noeud[0]]=noeud[1]
print(colorNodes)
# Pour colorer les noeuds, le format doit être une liste :
colorList=[colorNodes[node] for node in colorNodes]
print(colorList)
# Etiquetter les noeuds : le format doit être un dictionnaire :
liste = list(G.nodes(data='label'))
labels_nodes = {}
for noeud in liste:
    labels_nodes[noeud[0]]=noeud[1]
print(labels_nodes)
```

Traitement des sommets :

```
# Pour les étiquettes des arrêtes, le format doit être un dictionnaire :
labels_edges = {}
labels_edges = {edge:G.edges[edge]['weight'] for edge in G.edges}
print(labels_edges)
```

Une autre rédaction si celle ci-dessus pose problème :

```
# Pour les étiquettes des arrêtes , le format doit être un dictionnaire :
liste_edge = list(G.edges(data ='weight'))
labels_edges = {}
for edge in liste_edge :
    labels_edges[edge[0],edge[1]]= edge[1]
print(labels_edges)
```

Représentation du graphe :

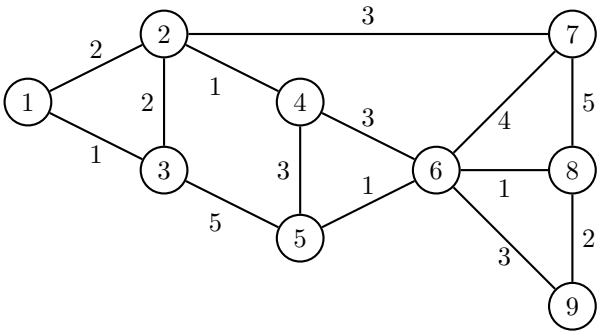
```
# Positions des sommets :
pos = nx.spring_layout(G)
# Sommets :
nx.draw_networkx_nodes(G, pos, node_size=700,node_color=colorList,alpha=0.9)
# Etiquettes des sommets :
nx.draw_networkx_labels(G, pos, labels=labels_nodes, \
                        with_labels=True,font_size=20, \
                        font_color='black', \
                        font_family='sans-serif')
# Arrêtes :
nx.draw_networkx_edges(G, pos,width=1)
nx.draw_networkx_edge_labels(G, pos, width=1, edge_labels=labels_edges,
font_color='red')
plt.axis('off')
plt.show()
```

Question 1
Expliquez en détail ce que fait chacune des parties de ces scripts.

Exercice 4
Pour éviter de devoir retaper tout ce script à chaque fois que l'on voudra l'utiliser, écrire une fonction **representation** prenant en entrée un graphe G et le représentant sous la même forme que le graphe présent dans l'exercice 5 (pas de couleur, sommets numérotés et poids des arrêtes affichés).

2 Recherche d'un arbre couvrant de poids minimal

Exercice 5
On considère le graphe G suivant :



1. Représenter ce graphe en faisant apparaitre les étiquettes des sommets et les poids des arrêtes (on pourra utiliser la fonction **representation** de l'exercice précédent).
2. Écrire une fonction **testnonisole** prenant en argument un graphe G et renvoyant True si le graphe G ne possède pas de sommet isolé et False si le graphe G possède un sommet isolé (c'est-à-dire relié à aucun autre sommet). *Indication :* On pourra utiliser la fonction **nx.degree**.
3. Expliquer ce que fait la fonction **nx.cycle_basis**.
4. Écrire une fonction **AlgoKruskal** prenant en entrée un graphe G et retournant un arbre couvrant de poids minimal ainsi que son poids. *Indication :* On pourra utiliser la fonction **nx.is_connected(G)**.
5. Appliquer cette fonction au graphe G précédent et représenter l'arbre couvrant de poids minimal obtenu à l'aide de la fonction **representation** de l'exercice précédent.

3 Algorithme de Dijkstra

Exercice 6
On s'intéresse pour finir à la recherche de plus courts chemins.

1. Écrire une fonction **AlgoDijktra** prenant en entrée un graphe G et retournant les plus courts chemins partant de chaque sommet de G et allant vers chacun des autres sommets de G ainsi que leurs longueurs.
2. Appliquer cette fonction au graphe G de l'exercice précédent et représenter à chaque fois le chemin le plus court à l'aide de la fonction **representation** de l'exercice 4.